1）实验平台：正点原子STM32mini开发板
2）摘自《正点原子STM32 不完全手册(HAL
库版)》关注官方微信号公众号，获取更多资料：正点原子

图 41.1.1 两个任务使用事件进行通信的示意图

在图 41.1.1 中任务 1 是发信方，任务 2 是收信方。任务 1
负责把信息发送到事件上，这项

操作叫做发送事件。任务 2
通过读取事件操作对事件进行查询：如果有信息则读取，否则等待。

读事件操作叫做请求事件。

为了把描述事件的数据结构统一起来，UCOSII
使用叫做事件控制块(ECB)的数据结构来描

述诸如信号量、邮箱（消息邮箱）和消息队列这些事件。事件控制块中包含包括等待任务表在

内的所有有关事件的数据，事件控制块结构体定义如下：

```
typedef struct

{

INT8U OSEventType;

//事件的类型

INT16U OSEventCnt;

//信号量计数器

void *OSEventPtr;
```

//消息或消息队列的指针

INT8U OSEventGrp;

//等待事件的任务组

INT8U OSEventTbl[OS_EVENT_TBL_SIZE];//任务等待表

#if OS_EVENT_NAME_EN > 0u

INT8U *OSEventName;

//事件名

#endif

} OS_EVENT;

信号量

信号量是一类事件。使用信号量的最初目的，是为了给共享资源设立一个标志，该标志表

示该共享资源的占用情况。这样，当一个任务在访问共享资源之前，就可以先对这个标志进行

查询，从而在了解资源被占用的情况之后，再来决定自己的行为。

信号量可以分为两种：一种是二值型信号量，另外一种是 N 值信号量。

二值型信号量好比家里的座机，任何时候，只能有一个人占用。而 N 值信号量，则好比公

共电话亭，可以同时有多个人（N 个）使用。

UCOSII 将二值型信号量称之为也叫互斥型信号量，将 N 值信号量称之为计数型信号量，

也就是普通的信号量。本章，我们介绍的是普通信号量，互斥型信号量的介绍，请

参考《嵌入

式实时操作系统 UCOSII 原理及应用》5.4 节。

接下来我们看看在 UCOSII 中，与信号量相关的几个函数（未全部列出，下同）。

1） 创建信号量函数

在使用信号量之前，我们必须用函数 OSSemCreate
来创建一个信号量，该函数的原型

为：OS_EVENT *OSSemCreate (INT16U
cnt)。该函数返回值为已创建的信号量的指针，而

参数 cnt 则是信号量计数器（OSEventCnt）的初始值。

2） 请求信号量函数

任务通过调用函数 OSSemPend 请求信号量，该函数原型如下：void
OSSemPend

(OS_EVENT *pevent, INT16U timeout, INT8U *err)。其中，参数 pevent
是被请求信号量的

指针，timeout 为等待时限，err 为错误信息。

为防止任务因得不到信号量而处于长期的等待状态，函数 OSSemPend
允许用参数

timeout 设置一个等待时间的限制，当任务等待的时间超过 timeout
时可以结束等待状态而

进入就绪状态。如果参数 timeout 被设置为 0，则表明任务的等待时间为无限长。

3） 发送信号量函数

任务获得信号量，并在访问共享资源结束以后，必须要释放信号量，释放信号量也
叫

做发送信号量，发送信号通过 OSSemPost 函数实现 。OSSemPost
函数在对信号量的计数

器操作之前，首先要检查是否还有等待该信号量的任务。如果没有，就把信号量计数器

OSEventCnt 加一；如果有，则调用调度器 OS_Sched(
)去运行等待任务中优先级别最高的

任务。函数 OSSemPost 的原型为：INT8U OSSemPost(OS_EVENT
*pevent)。其中，pevent

为信号量指针，该函数在调用成功后，返回值为
OS_ON_ERR，否则会根据具体错误返回

OS_ERR_EVENT_TYPE、OS_SEM_OVF。

4） 删除信号量函数

应用程序如果不需要某个信号量了，那么可以调用函数 OSSemDel
来删除该信号量，

该函数的原型为：OS_EVENT *OSSemDel (OS_EVENT *pevent,INT8U opt,
INT8U *err)。

其中，pevent 为要删除的信号量指针，opt 为删除条件选项，err 为错误信息。

邮箱

在多任务操作系统中，常常需要在任务与任务之间通过传递一个数据（这种数据叫做“消

息” ）的方式来进行通信。为了达到这个目的，可以在内存中创建一个存储空间作为该数据的

缓冲区。如果把这个缓冲区称之为消息缓冲区，这样在任务间传递数据（消息）的最简单办法

就是传递消息缓冲区的指针。我们把用来传递消息缓冲区指针的数据结构叫做邮箱

（消息邮箱）。

在 UCOSII 中，我们通过事件控制块的 OSEventPrt
来传递消息缓冲区指针，同时使事件控

制块的成员 OSEventType 为常数
OS_EVENT_TYPE_MBOX，则该事件控制块就叫做消息邮箱。

接下来我们看看在 UCOSII 中，与消息邮箱相关的几个函数。

1）创建邮箱函数

创建邮箱通过函数 OSMboxCreate 实现，该函数原型为：OS_EVENT
*OSMboxCreate

(void *msg)。函数中的参数 msg
为消息的指针，函数的返回值为消息邮箱的指针。

调用函数 OSMboxCreate 需先定义 msg
的初始值。在一般的情况下，这个初始值为

NULL；但也可以事先定义一个邮箱，然后把这个邮箱的指针作为参数传递到函数

OSMboxCreate 中，使之一开始就指向一个邮箱。

2）向邮箱发送消息函数

任务可以通过调用函数 OSMboxPost
向消息邮箱发送消息，这个函数的原型为：INT8U

OSMboxPost (OS_EVENT *pevent,void *msg)。其中 pevent
为消息邮箱的指针，msg 为消息

指针。

3）请求邮箱函数

当一个任务请求邮箱时需要调用函数
OSMboxPend，这个函数的主要作用就是查看邮

箱指针 OSEventPtr 是否为 NULL，如果不是 NULL
就把邮箱中的消息指针返回给调用函数

的任务，同时用 OS_NO_ERR 通过函数的参数 err
通知任务获取消息成功；如果邮箱指针

OSEventPtr 是 NULL，则使任务进入等待状态，并引发一次任务调度。

函数 OSMboxPend 的原型为：void *OSMboxPend (OS_EVENT *pevent,
INT16U timeout,

INT8U *err)。其中 pevent 为请求邮箱指针，timeout 为等待时限，err
为错误信息。

4 ）查询邮箱状态函数

任务可以通过调用函数 OSMboxQuery
查询邮箱的当前状态。该函数原型为：INT8U

OSMboxQuery(OS_EVENT *pevent,OS_MBOX_DATA *pdata)。其中 pevent
为消息邮箱指

针，pdata 为存放邮箱信息的结构。

5 ）删除邮箱函数

在邮箱不再使用的时候，我们可以通过调用函数 OSMboxDel
来删除一个邮箱，该函

数原型为：OS_EVENT *OSMboxDel(OS_EVENT *pevent,INT8U opt,INT8U
*err)。其中

pevent 为消息邮箱指针，opt 为删除选项，err 为错误信息。

关于 UCOSII
信号量和邮箱的介绍，就到这里。更详细的介绍，请参考《嵌入式实时操作

系统 UCOSII 原理及应用》第五章。

41.2 硬件设计

本节实验功能简介：本章我们在 UCOSII 里面创建 6
个任务（不含统计任务和空闲任务）：

开始任务、LED0 任务、LED1
任务、触摸屏任务、主任务和按键扫描任务，开始任务用于创建

信号量、创建邮箱、初始化统计任务以及其他任务的创建，之后挂起；LED0
任务用于 DS0 控

制，提示程序运行状况；LED1
任务用于测试信号量，通过请求信号量函数，每得到一个信号

量，DS1 就亮一下；触摸屏任务用于在屏幕上画图，可以用于测试 CPU
使用率；按键扫描任

务用于按键扫描，优先级最高，将得到的键值通过消息邮箱发送出去；主任务则通过查询消息

邮箱获得键值，并根据键值执行信号量发送（DS1
控制）、触摸区域清屏和触摸屏校准等控制。

所要用到的硬件资源如下：

1）指示灯 DS0 、DS1

2）三个按键（KEY0/KEY1/WK_UP）

3）TFTLCD 模块

这些，我们在前面的学习中都已经介绍过了。

41.3 软件设计

本章，我们在第二十六章实验（实验 21 ）的基础上修改。首先，是 UCOSII
代码的添加，

具体方法同上一章一模一样，本章就不再详细介绍了。不过，本章我们将

OS_TICKS_PER_SEC

设置为 500，即 UCOSII 的时钟节拍为 2ms。

在加入 UCOSII 代码后，我们只需要修改 test.c 函数了，打开 main.c，输入如下代码：

```
//START 任务

//设置任务优先级

#define START_TASK_PRIO

10 //开始任务的优先级为最低

//设置任务堆栈大小

#define START_STK_SIZE

128

//任务任务堆栈

OS_STK START_TASK_STK[START_STK_SIZE];

//任务函数

void start_task(void *pdata);

//触摸屏任务

//设置任务优先级

#define TOUCH_TASK_PRIO 7

//设置任务堆栈大小

#define TOUCH_STK_SIZE 128
```

```
//任务堆栈
OS_STK TOUCH_TASK_STK[TOUCH_STK_SIZE];
//任务函数
void touch_task(void *pdata);
//LED0 任务
//设置任务优先级
#define LED0_TASK_PRIO
6
//设置任务堆栈大小
#define LED0_STK_SIZE
128
//任务堆栈
OS_STK LED0_TASK_STK[LED0_STK_SIZE];
//任务函数
void led0_task(void *pdata);
//LED01 任务
//设置任务优先级
#define LED1_TASK_PRIO
5
//设置任务堆栈大小
```

#define LED1_STK_SIZE

128

//任务堆栈

OS_STK LED1_TASK_STK[LED1_STK_SIZE];

//任务函数

void led1_task(void *pdata);

//主任务

//设置任务优先级

#define MAIN_TASK_PRIO 4

//设置任务堆栈大小

#define MAIN_STK_SIZE 128

//任务堆栈

OS_STK MAIN_TASK_STK[MAIN_STK_SIZE];

//任务函数

void main_task(void *pdata);

//按键扫描任务

//设置任务优先级

#define KEY_TASK_PRIO

3

//设置任务堆栈大小

```
#define KEY_STK_SIZE 128

//创建任务堆栈空间

OS_STK KEY_TASK_STK[KEY_STK_SIZE];

//任务函数接口

void key_task(void *pdata);

OS_EVENT * msg_key;

//按键邮箱事件块指针

OS_EVENT * sem_led1;

//LED1 信号量指针

//加载主界面

void ucos_load_main_ui(void)

{

LCD_Clear(WHITE); //清屏

POINT_COLOR=RED; //设置字体为红色

LCD_ShowString(30,10,200,16,16,"Mini STM32");

LCD_ShowString(30,30,200,16,16,"UCOSII TEST2");

LCD_ShowString(30,50,200,16,16,"ATOM@ALIENTEK");

LCD_ShowString(30,75,200,16,16,"KEY0:LED1 KEY_UP:ADJUST");

LCD_ShowString(30,95,200,16,16,"KEY1:CLEAR");

LCD_ShowString(80,210,200,16,16,"Touch Area");
```

```
LCD_DrawLine(0,120,lcddev.width,120);

LCD_DrawLine(0,70,lcddev.width,70);

LCD_DrawLine(150,0,150,70);

POINT_COLOR=BLUE;//设置字体为蓝色

LCD_ShowString(160,30,200,16,16,"CPU: %");

LCD_ShowString(160,50,200,16,16,"SEM:000");

}

int main(void)

{

HAL_Init(); //初始化 HAL 库

Stm32_Clock_Init(RCC_PLL_MUL9); //设置时钟,72M

delay_init(72);

//初始化延时函数

uart_init(115200);

//初始化 USART

LED_Init();

//初始化 LED

KEY_Init(); //初始化按键

LCD_Init(); //初始化 LCD

tp_dev.init();
```

```
//初始化触摸屏
ucos_load_main_ui();
//加载主界面
OSInit();
//初始化 UCOSII
OSTaskCreateExt((void(*)(void*) )start_task, //任务函数
(void* )0, //传递给任务函数的参数
(OS_STK* )&START_TASK_STK[START_STK_SIZE-1],//任务堆栈栈顶
(INT8U )START_TASK_PRIO, //任务优先级
(INT16U )START_TASK_PRIO, //任务 ID，这里设置为和优先级一样
(OS_STK* )&START_TASK_STK[0], //任务堆栈栈底
(INT32U )START_STK_SIZE, //任务堆栈大小
(void* )0, //用户补充的存储区
(INT16U )OS_TASK_OPT_STK_CHK|
OS_TASK_OPT_STK_CLR|OS_TASK_OPT_SAVE_FP);
//任务选项,为了保险起见，所有任务都保存浮点寄存器的值
OSStart(); //开始任务
}
//画水平线
//x0,y0:坐标
```

```
//len:线长度

//color:颜色

void gui_draw_hline(u16 x0,u16 y0,u16 len,u16 color)

{

if(len==0)return;

LCD_Fill(x0,y0,x0+len-1,y0,color);

}

//画实心圆

//x0,y0:坐标

//r:半径

//color:颜色

void gui_fill_circle(u16 x0,u16 y0,u16 r,u16 color)

{

u32 i;

u32 imax = ((u32)r*707)/1000+1;

u32 sqmax = (u32)r*(u32)r+(u32)r/2;

u32 x=r;

gui_draw_hline(x0-r,y0,2*r,color);

for (i=1;i<=imax;i++)

{
```

```
if ((i*i+x*x)>sqmax)// draw lines from outside
{
if (x>imax)
{
gui_draw_hline (x0-i+1,y0+x,2*(i-1),color);
gui_draw_hline (x0-i+1,y0-x,2*(i-1),color);
}
x--;
}
// draw lines from inside (center)
gui_draw_hline(x0-x,y0+i,2*x,color);
gui_draw_hline(x0-x,y0-i,2*x,color);
}
}
//两个数之差的绝对值
//x1,x2：需取差值的两个数
//返回值：|x1-x2|
u16 my_abs(u16 x1,u16 x2)
{
if(x1>x2)return x1-x2;
```

```
else return x2-x1;

}

//画一条粗线

//(x1,y1),(x2,y2):线条的起始坐标

//size：线条的粗细程度

//color：线条的颜色

void lcd_draw_bline(u16 x1, u16 y1, u16 x2, u16 y2,u8 size,u16 color)

{

u16 t;

int xerr=0,yerr=0,delta_x,delta_y,distance;

int incx,incy,uRow,uCol;

if(x1<size|| x2<size||y1<size|| y2<size)return;

delta_x=x2-x1; //计算坐标增量

delta_y=y2-y1;

uRow=x1;

uCol=y1;

if(delta_x>0)incx=1; //设置单步方向

else if(delta_x==0)incx=0;//垂直线

else {incx=-1;delta_x=-delta_x;}

if(delta_y>0)incy=1;
```

```
else if(delta_y==0)incy=0;//水平线

else{incy=-1;delta_y=-delta_y;}

if( delta_x>delta_y)distance=delta_x; //选取基本增量坐标轴

else distance=delta_y;

for(t=0;t<=distance+1;t++ )//画线输出

{

gui_fill_circle(uRow,uCol,size,color);//画点

xerr+=delta_x ;

yerr+=delta_y ;

if(xerr>distance)

{

xerr-=distance;

uRow+=incx;

}

if(yerr>distance)

{

yerr-=distance;

uCol+=incy;

}

}
```

```
}
//开始任务
void start_task(void *pdata)
{
OS_CPU_SR cpu_sr=0;
pdata = pdata;
msg_key=OSMboxCreate((void*)0); //创建消息邮箱
sem_led1=OSSemCreate(0);
//创建信号量
OSStatInit();
//初始化统计任务.这里会延时 1 秒钟左右
OS_ENTER_CRITICAL();
//进入临界区(无法被中断打断)
OSTaskCreateExt((void(*)(void*) )touch_task,
(void* )0,
(OS_STK* )&TOUCH_TASK_STK[TOUCH_STK_SIZE-1],
(INT8U )TOUCH_TASK_PRIO,
(INT16U )TOUCH_TASK_PRIO,
(OS_STK* )&TOUCH_TASK_STK[0],
(INT32U )TOUCH_STK_SIZE,
```

```
(void* )0,
(INT16U )OS_TASK_OPT_STK_CHK|
OS_TASK_OPT_STK_CLR|OS_TASK_OPT_SAVE_FP);
...创建任务(详细请看源码)...
OS_EXIT_CRITICAL();
//退出临界区(可以被中断打断)
}
//LED0 任务
void led0_task(void *pdata)
{
u8 t;
while(1)
{
t++; delay_ms(10);
if(t==8)LED0=1;
//LED0 灭
if(t==100) { t=0; LED0=0;} //LED0 亮
}
}
//LED1 任务
```

```
void led1_task(void *pdata)
{
u8 err;
while(1)
{
OSSemPend(sem_led1,0,&err);
LED1=0;delay_ms(200);
LED1=1; delay_ms(800);
}
}
//触摸屏任务
void touch_task(void *pdata)
{
while(1)
{
tp_dev.scan(0);
if(tp_dev.sta&TP_PRES_DOWN)
//触摸屏被按下
{
if(tp_dev.x[0]<lcddev.width&&tp_dev.y[0]<lcddev.height&&tp_dev.y[0]>12
```

```
0)
{
TP_Draw_Big_Point(tp_dev.x[0],tp_dev.y[0],RED);
//画图
delay_ms(2);
}
}else delay_ms(10);
//没有按键按下的时候
}
}
//主任务
void main_task(void *pdata)
{
u32 key=0; u8 err; u8 semmask=0;u8 tcnt=0;
while(1)
{
key=(u32)OSMboxPend(msg_key,10,&err);
switch(key)
{
case KEY0_PRES://发送信号量
```

```
semmask=1;
OSSemPost(sem_led1);
break;
case KEY1_PRES://清除
LCD_Fill(0,121,lcddev.width,lcddev.height,WHITE);
break;
case WKUP_PRES://校准
OSTaskSuspend(TOUCH_TASK_PRIO); //挂起触摸屏任务
if((tp_dev.touchtype&0X80)==0)TP_Adjust();
OSTaskResume(TOUCH_TASK_PRIO);
//解挂
ucos_load_main_ui();
//重新加载主界面
break;
}
if(semmask||sem_led1->OSEventCnt)//需要显示 sem
{
POINT_COLOR=BLUE;
LCD_ShowxNum(192,50,sem_led1->OSEventCnt,3,16,0X80);//显示信号量值
if(sem_led1->OSEventCnt==0)semmask=0; //停止更新
```

```
}
if(tcnt==50)//0.5 秒更新一次 CPU 使用率
{
tcnt=0;
POINT_COLOR=BLUE;
LCD_ShowxNum(192,30,OSCPUUsage,3,16,0); //显示 CPU 使用率
}
tcnt++; delay_ms(10);
}
}
//按键扫描任务
void key_task(void *pdata)
{
u8 key;
while(1)
{
key=KEY_Scan(0);
if(key)OSMboxPost(msg_key,(void*)key);//发送消息
delay_ms(10);
}
```

}

该部分代码我们创建了 6
个任务：start_task、led0_task、touch_task、led1_task、main_task

和 key_task，优先级分别是 10 和 7~3，堆栈大小都是是 128。

该程序的运行流程就比上一章复杂了一些，我们创建了消息邮箱
msg_key，用于按键任务

和主任务之间的数据传输（传递键值），另外创建了信号量 sem_led1，用于
LED1 任务和主任

务之间的通信。

本代码中，我们使用了 UCOSII 提供的 CPU 统计任务，通过 OSStatInit 初始化
CPU 统计任

务，然后在主任务中显示 CPU 使用率。

另外，在主任务中，我们用到了任务的挂起和恢复函数，在执行触摸屏校准的时候
，我们

必须先将触摸屏任务挂起，待校准完成之后，再恢复触摸屏任务。这是因为触摸屏
校准和触摸

屏任务都用到了触摸屏和
TFTLCD，而这两个东西是不支持多个任务占用的，所以必须采用独

占的方式使用，否则可能导致数据错乱。

软件设计部分就为大家介绍到这里。

41.4 下载验证

在代码编译成功之后，我们通过下载代码到 MiniSTM32 开发板上，可以看到 LCD
显示界

面如图 41.4.1 所示：